On the Performance of Flooding-Based Resource Discovery

Vassilios V. Dimakopoulos and Evaggelia Pitoura

Department of Computer Science, University of Ioannina
P.O. Box 1186, Ioannina, Greece GR-45110
Tel: +30 26510 {98809, 98811}, Fax: +30 26510 98890
E-mail: [dimako,pitoura]@cs.uoi.gr

Abstract

We consider flooding-based resource discovery in distributed systems. With flooding, a node searching for a resource contacts its neighbors in the network, which in turn contact their own neighbors and so on until a node possessing the requested resource is located. Flooding assumes no knowledge about the network topology or the resource distribution thus offering an attractive means for resource discovery in dynamically evolving networks such as peer-to-peer systems. We provide analytical results for the performance of a number of flooding-based approaches that differ in the set of neighbors contacted at each step. The performance metrics we are interested in are the probability of locating a resource and the average number of steps and messages for doing so. We study both uniformly random resource requests and requests in the presence of popular (hot) resources. Our analysis is also extended to take into account the fact that nodes may become unavailable either due to failures or voluntary departures from the system. Our analytical results are validated through simulation.

Index terms: resource discovery, flooding, distributed systems, performance analysis, peer-to-peer systems, multi-agent systems

1 Introduction

In a distributed resource sharing system, several nodes offer resources (such as data items, programs, computational power, or web services) that other nodes want to use. Popular examples of resource sharing systems include peer-to-peer (p2p) networks, computational grids, multi-agent systems, and the web. An issue central to all such systems is how to locate a resource, that is, a node in the distributed system that provides it. There are various approaches to resource discovery. Centralized approaches utilize directories that maintain mappings between the resources and the nodes that offer them. However, directory servers constitute performance bottlenecks as well as single points of failure. Moreover, centralized solutions do not scale well in dynamic systems in which nodes enter and leave the system asynchronously thus creating large update volumes.

In this paper, we consider decentralized resource discovery mechanisms based on flooding. With flooding, a node that wants a particular resource contacts its neighbors in the system, which in turn contact their own neighbors until a node that provides the requested resource is reached. Flooding enables resource discovery without directories or knowledge of the specific topology of the system, thus, offering an attractive mechanism for resource discovery in dynamically evolving networks. For example, Gnutella [8], a popular peer-to-peer file sharing system, utilizes some form of flooding-based discovery.

In abstract terms, we assume a distributed system with N nodes where each node provides a number of resources. There are R different types of resources. Each node knows about d other nodes which are called its *neighbors*. The system is modeled as a directed graph $G(V, E)$ where each node of the graph corresponds to a node of the distributed system and there is an edge from node A to each of A's neighbors. Because each edge in G may not correspond to a single physical link, graph G is called the *overlay network*. There is no knowledge about the size of the network. An example is shown in Figure 1.

In addition, we assume that each node has a local cache of size k, where it stores information about k different resources, that is, for each of the k resources the contact information of one node that provides it. The goal of caching is to exploit locality in resource requests that has been reported in many resource sharing systems (such as in peer-to-peer systems [21, 16] and the web [7]).

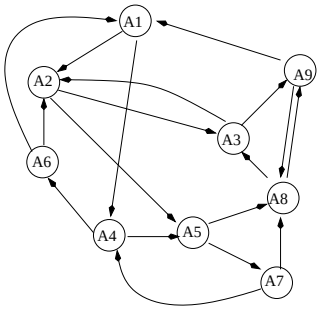

Figure 1: An overlay network with $d = 2$.

Given an overlay network, the resource discovery problem is the following: how can a node A that needs a particular type of resource x, find a node that provides it. In pure flooding, node A initially searches its own cache. If it finds the resource there, it extracts the corresponding contact information and the search ends. If resource x is not found in the local cache, A sends a message querying all its neighbors, which in turn propagate the message to all their own neighbors and so on. Since, pure flooding overwhelms the network with search messages, we consider variations that restrict the search space, such as random walkers or paths [15, 6] and probabilistic flooding or teeming [3, 6, 11] that contact subsets of each node's neighbors at each step.

Although, there has been a lot of empirical studies (e.g. [19]) and some simulation-based analysis (e.g. [15]) of flooding and its variants, analytical results are lacking. In this paper, we provide analytical estimations of the performance of flooding-based search and its variations. We validate our analytical estimations through simulation. We study both uniformly random requests and requests when there exist hot spots. In the former case, we assume that the entries in the caches are random, that is, the entries of each cache is a uniformly random subset of the available resources. In the latter case, we assume that some resources (i.e., the hot spots) appear in a large number of caches. This is motivated by the fact that in many settings, some resources are more popular than others (for instance, in peer-to-peer systems [21]); this results in hot resources being cached more often. In addition, we extend our analysis to take into account node unavailability. Node unavailability may be the result of node failures or of the dynamic nature of some resource sharing systems where nodes leave and enter the system at will.

The performance metrics we are interested in are:

- the probability of locating a resource within t steps,
- the average number of steps needed to locate a resource, and
- the average number of message transmissions required.

The remainder of this paper is structured as follows. The flooding-based search procedures are introduced in Section 2 and analytical results about their performance are derived in Section 3. Next, our performance results are extended for accounting for hot spots (in Section 4) and node unavailability (in Section 5). In Section 6, we present related research and various applications of flooding-based search. In Section 7, we summarize our results and outline our plans for future work.

2 Search algorithms

We assume an overlay network where each node has d neighbors and maintains a local cache with k entries. When a node A needs a particular type of resource x, it initially searches its own cache. If it finds the resource there, it extracts the corresponding contact information and the search ends. If resource x is not found in the local cache, A sends a message querying *all* or a *subset* of its neighbors, which in turn propagate the message to their neighbors and so on.

To avoid overwhelming the network with search requests, search is limited to a maximum number of steps, t (similar to the Time To Live (TTL) parameter in many network protocols). In particular, the search message contains a counter field initialized to t. Any intermediate node that receives the message first decrements the counter by 1. If the counter value is not 0, the node proceeds as normal; otherwise the node does not contact its neighbors and sends a positive (negative) response to the inquiring node if x is found (not found) in its cache.

When the search ends, the inquiring node A will either have the contact information for resource x or a set of negative responses. In the latter case, node A assumes that a node offering the resource cannot be found. Note that we make no assumption about network connectivity. Disconnectedness may indeed occur because the network is dynamic. This is quite possible in peer-to-peer systems. For example, measurement

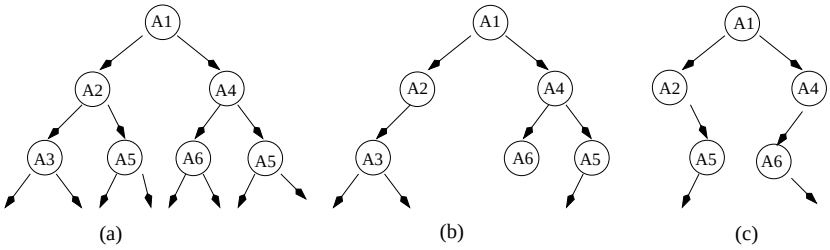

Figure 2: Searching the overlay network of Figure 1: (a) flooding, (b) teeming, (c) random paths ($p = 2$)

studies have shown several disjoint Gnutella overlay networks existing simultaneously in the Internet [19].

We consider three different search strategies based on what subset of its neighbors each node contacts, namely the flooding, teeming and random paths strategies.

2.1 Flooding

With (pure) flooding, node A that searches for a resource x checks its cache, and if the resource is not found there, A contacts *all* its neighbors. In turn, A's neighbors check their caches and if the resource is not found locally, they propagate the search message to *all* their neighbors. The procedure ends when either the resource is found or a maximum of t steps is reached. The scheme, in essence, broadcasts the inquiring message.

As the search progresses, a d-ary tree is unfolded rooted at the inquiring node A. An example is shown in Figure 2(a). Note that the term "tree" is not accurate in graph-theoretic terms since a node may be contacted by two or more other nodes, but we will use it here as it helps to visualize the situation. This search tree has (at most) d^i different nodes at the ith level, $i \geq 0$, which means that at the ith step of the search algorithm, there will be (at most) d^i different nodes contacted.

2.2 Teeming

To reduce the number of messages, we consider a variation of flooding called *teeming* [6] probabilistic flooding [3] or random BFS [11]. At each step, if the resource is not

found in the local cache of a node, the node propagates the inquiring message only to a *random subset* of its neighbors. We denote by ϕ the fixed probability of selecting a particular neighbor. In contrast with flooding, the search tree is not a d-ary one any more (Figure 2(b)). A node in the search tree may have between 0 to d children, $d\phi$ being the average case. Flooding can be seen as a special case of teeming for which $\phi = 1$.

2.3 Random paths

Although, depending on ϕ, teeming can reduce the overall number of messages, both teeming and flooding suffer from an exponential number of messages. One approach to eliminate this drawback is performing a random path or random walker [6, 15] search as follows: each node contacts only one of its neighbors (randomly). The search space formed ends up being a single random path in the overlay network. This scheme propagates one single message along the path and the inquiring node will be expecting one single answer.

This scheme is generalized as follows: the root node (i.e., the inquiring node A) constructs $p \geq 1$ random paths. In particular, if x is not in its cache, A asks p out of its d neighbors (not just one of them). All the other (intermediate) nodes construct a simple path as above, by asking exactly one of their neighbors. This way, we end up with p different paths unfolding concurrently (Figure 2(c)). The search algorithm produces less messages than flooding or teeming but needs more steps to locate a resource.

3 Performance analysis

In this section, we analyze the performance of the three search algorithms presented in the previous section. In particular, we will assume that the algorithms operate for a maximum of t steps and we will derive analytically three important performance measures:

- The probability, Q_t, that the resource we are looking for is found within the t steps; Q_t should be as high as possible.

- The average number of steps, $\overline{S_t}$, needed for locating a resource (given that the resource is found), which should be kept low.

- The average number of message transmissions, $\overline{M_t}$, occurring during the course of the algorithm. Efficient strategies should require as few messages as possible in order not to saturate the underlying network resources (which, however, may lead to a higher number of steps).

3.1 Preliminaries

Appendix A contains a summary of the notation we will use. We assume a random directed network of N nodes, each node having d links to d other nodes, called its neighbors. Each node is equipped with a cache of size k storing the resource-provider pairs it knows of. A node may also offer resources itself. There are in total R different resources provided collectively in the network. Some node ("inquiring node") will be assumed to search for some resource x, and we let n_x be the number of nodes offering that particular resource. Finally, the caches are assumed to be in steady-state, all caches being full, meaning that each node knows of exactly k resources (along with their providers).

Given a resource x, assume that the probability that x is known to a particular node is equal to F_1. A resource is known to a node when the node either offers it or has cached one of its providers.

Now let us denote by Q_t the probability of locating x within t steps of an algorithm. An important performance measure is the average number of steps needed to find a resource x. Given that a resource is located within t steps, the average number of steps is shown to be:

$$\overline{S_t} = t - \frac{1}{Q_t} \sum_{i=1}^{t-1} Q_i. \tag{1}$$

The derivation of (1) is given at the end of this section for presentation purposes.

To derive the performance measures, we need to calculate F_1 and Q_t. F_1 is dependent only on the resource distribution and the cache contents, while Q_t also depends on the particular search strategy utilized. We compute F_1 next, and we defer the derivations of Q_t for each strategy to the corresponding subsections.

In what follows, we assume that the content of each cache is completely random; in other words a node's k cached resources are a uniformly random subset of the R available resources. We consider other distributions for the cache contents in later sections.

For a node to not know about a resource x, it must not provide it and it must not have it cached. Thus,

$$F_1 = 1 - P[\text{node does not offer } x]P[x \notin \text{node's cache}]$$

Since resource x is offered by n_x nodes in total, we have:

$$P[\text{node offers } x] = \frac{n_x}{N}.$$

The number of ways to choose k elements out of a set of R elements so that a particular element is not chosen is $\binom{R-1}{k}$. Since the k elements of the node's cache are assumed completely random, we obtain:

$$P[x \notin \text{node's cache}] = \frac{\binom{R-1}{k}}{\binom{R}{k}} = \frac{R-k}{R},$$

which, gives:

$$F_1 = 1 - \left(1 - \frac{n_x}{N}\right)\left(\frac{R-k}{R}\right) = 1 - \frac{(N-n_x)(R-k)}{NR}.$$

In what follows, we let $a = (N-n_x)(R-k)/NR$, so that $F_1 = 1 - a$.

Derivation of Eq. (1) Let s_i denote the probability of locating the resource at exactly the ith step of an algorithm. Then, the probability of locating it within $t \geq 1$ steps is given by:

$$Q_t = \sum_{i=0}^{t} s_i,$$

which means that $s_t = Q_t - Q_{t-1}$. Given that the resource is found within t steps, the probability of locating it exactly at the ith step is equal to s_i/Q_t and the average number of steps is equal to:

$$\overline{S_t} = \frac{1}{Q_t}\sum_{i=1}^{t} i s_i = \frac{1}{Q_t}\left(\sum_{i=1}^{t-1} i s_i + t s_t\right) = \frac{1}{Q_t}(Q_{t-1}\overline{S_{t-1}} + t s_t).$$

Substituting for s_t, we obtain:

$$Q_t\overline{S_t} = Q_{t-1}\overline{S_{t-1}} + tQ_t - tQ_{t-1}.$$

Letting $g_t = Q_t\overline{S_t}$, the recurrence takes the following form:

$$g_t = g_{t-1} + tQ_t - tQ_{t-1},$$

with a boundary of $g_0 = S_0Q_0 = 0$. It is easy to see that,

$$g_t = \sum_{i=1}^{t} iQ_i - \sum_{i=1}^{t} iQ_{i-1} = tQ_t - \sum_{i=0}^{t-1} Q_i,$$

and, since $g_t = Q_t\overline{S_t}$, (1) follows.

3.2 Performance of flooding

With flooding, each node receiving the inquiring message transmits it to all its neighbors (unless it offers the required resource x or x is in its cache). As the algorithm progresses, a d-ary "tree" is unfolded rooted at the inquiring node. This search tree has (at most) d^i different nodes in the ith level, $i \geq 0$, which means that at the ith step of the algorithm, there will be (at most) d^i different nodes contacted.*

If the requested resource is not found, it is because the inquiring node did not know about it (with probability $a = 1 - F_1$) and because none of the d "subtrees" unfolding from the inquiring node's neighbors replied with a positive answer. Such a subtree has $t - 1$ levels; it sends an affirmative reply only if it locates the requested resource (with probability Q_{t-1}). Thus, the probability of not finding x is given by the following recurrence:

$$1 - Q_t^{(F)} = a\left(1 - Q_{t-1}^{(F)}\right)^d,$$

where F is used to denote the flooding algorithm. The boundary condition is, clearly, $Q_0^{(F)} = F_1 = 1 - a$. Notice that, as discussed above, the d subtrees of the inquiring node may not be disjoint and as a consequence the events of locating the resource in the d subtrees may not be statistically independent. The above recurrence is an approximation

*Since the network is assumed random, a node may be contacted through more than one path. This fact may limit the number of different nodes in the ith level of the tree to less that d^i.

which overestimates somewhat the probability but it simplifies the analysis and does not introduce significant error as indicated by our simulation results.

Setting $q_t = 1 - Q_t^{(F)}$, the recurrence becomes $q_t = aq_{t-1}^d$ which easily evaluates to $q_t = a^{(d^{t+1}-1)/(d-1)}$, giving:

$$Q_t^{(F)} = 1 - a^{\frac{d^{t+1}-1}{d-1}} \tag{2}$$

The average number of steps needed to locate a resource can be found by substituting (2) into (1). After some straightforward manipulations, the average number of steps is found to be, for $t \geq 1$,

$$\overline{S_t^{(F)}} = t - \frac{t}{Q_t^{(F)}} + \frac{1}{Q_t^{(F)}} \sum_{i=1}^{t} a^{\frac{d^i-1}{d-1}}. \tag{3}$$

We know of no closed-form formula for the sum in (3).

Let us now compute the number of messages in the flooding algorithm. We will assume that nodes reply directly to the inquiring node. If the resource is found at the inquiring node there will be no message transmissions. Otherwise, there will be d transmissions to the d neighbors of the root, plus the transmissions internal to each of the d subtrees T rooted at those neighbors. Symbolically, we have:

$$\overline{M_t^{(F)}} = (1 - F_1)(d + dm(t-1)),$$

where $m(t-1)$ are the transmissions occurring within a particular subtree T with $t-1$ levels. For such a subtree T, if x is found in its root node there will be 1 positive reply back to the inquiring node; otherwise, there will be d message transmissions to the children of the root plus the transmissions inside the d subtrees T' (with $t-2$ levels) rooted at the node's children. This leads to the following recurrence:

$$\begin{aligned} m(t-1) &= 1 \times F_1 + (d + dm(t-2)) \times (1 - F_1) \\ &= adm(t-2) + ad + 1 - a, \end{aligned}$$

with a boundary condition of:

$$m(0) = 1, \tag{4}$$

since each of the last nodes receiving the message (at the tth step) will always reply to the inquiring node whether it knows x or not. The solution to the above recurrence is:

$$m(t-1) = (ad)^{t-1} + \frac{(ad)^{t-1} - 1}{ad - 1}(ad + 1 - a),$$

which gives, after some manipulation:

$$\overline{M_t^{(F)}} = a + \frac{(c^t - 1)(2c - a)}{c - 1}, \qquad c = ad,\ t \geq 1. \tag{5}$$

Eq. (5) shows that (as anticipated) the flooding algorithm requires an exponential number of messages with respect to the nodes' degree (d).

3.3 Performance of teeming

With teeming, a node propagates the inquiring message to each of its neighbors with a fixed probability ϕ. Again, if the requested resource x is not found, it is due to two facts: first, the inquiring node does not know about it (occurring with probability $1 - F_1$). Second, none of the d "subtrees" unfolding from the inquiring node's neighbors replies with a positive answer. Such a subtree has $t - 1$ levels; it sends an affirmative reply only if it asked by the inquiring node (with probability ϕ) and indeed locates the requested resource (with probability $Q_{t-1}^{(T)}$). Thus, the probability of not finding x is given by the following recurrence:

$$1 - Q_t^{(T)} = (1 - F_1)\left(1 - \phi Q_{t-1}^{(T)}\right)^d,$$

which gives:

$$Q_t^{(T)} = 1 - a\left(1 - \phi Q_{t-1}^{(T)}\right)^d. \tag{6}$$

where T is used to denote the teeming algorithm. The boundary condition is, clearly, $Q_0^{(T)} = F_1 = 1 - a$.

The average number of steps, for $t \geq 1$, can be found using (1):

$$\overline{S_t^{(T)}} = t - \frac{1}{Q_t^{(T)}} \sum_{i=0}^{t-1} Q_i^{(T)}. \tag{7}$$

The average number of messages is computed almost identically with the flooding case; the only difference is that since a node transfers the message to a particular child with probability ϕ, the average number of steps is given by:

$$\overline{M_t^{(T)}} = (1 - F_1)(d\phi + d\phi m(t - 1)),$$

where $m(t - 1)$ is the transmissions occurring within a particular subtree T with $t - 1$ levels. The recurrence (see Section 3.2) takes the form:

$$m(t - 1) = 1 \times F_1 + (d\phi + d\phi m(t - 2)) \times (1 - F_1),$$

which eventually gives:

$$\overline{M_t^{(T)}} = a + \frac{(c^t - 1)(2c - a)}{c - 1}, \qquad c = ad\phi,\ t \geq 1. \tag{8}$$

Teeming also requires an exponential number of messages, which however grows slower than in the case of flooding; its rate is controlled by the probability ϕ.

3.4 Performance of the random paths algorithm

When using the random paths algorithm, the inquiring node transmits the message to $p \geq 1$ of its neighbors. Each neighbor then becomes the root of a randomly unfolding path. There is a chance that those p paths meet at some node(s), thus they may not always be disjoint. However, for simplification purposes, we will assume that they are completely disjoint and thus statistically independent. This approximation introduces negligible error (especially if t is not large) as our experiments show.

The probability of not finding the required resource x is given by:

$$1 - Q_t^{(p)} = (1 - F_1)(1 - q_{t-1})^p,$$

where $a = 1 - F_1$ is the probability that the inquiring node does not know about x and $1 - q_{t-1}$ is the probability of not locating the resource in one of the p randomly unfolding paths of length $t - 1$. Such a path has t nodes and for failing to locate the resource, none of its nodes should know about x, which means that:

$$1 - q_{t-1} = (1 - F_1)^t = a^t,$$

yielding:

$$Q_t^{(p)} = 1 - a^{pt+1}. \tag{9}$$

Using (9) and (1), we can easily find the average number of steps:

$$\overline{S_t^{(p)}} = t - \frac{1}{Q_t^{(p)}} \sum_{i=0}^{t-1} (1 - a^{pi+1}),$$

which after some algebraic manipulations gives, for $t \geq 1$,

$$\overline{S_t^{(p)}} = \frac{a - (1 + t - ta^p)a^{pt+1}}{(1 - a^p)(1 - a^{pt+1})}. \tag{10}$$

Setting $p = 1$ the above formulas give the corresponding performance measures for the single-path algorithm.

Finally, the number of message transmissions can be calculated using arguments similar to the ones in Section 3.2. If x is not known by the inquiring node's cache, there will be p message transmissions to p of its children, plus the message transmissions in each of the p paths:

$$\overline{M_t^{(p)}} = (1 - F_1)(p + pm(t-1)),$$

where $m(t-1)$ is the transmissions occurring within a particular path P of $t-1$ nodes. For such a path P, if x is found in its root node there will be 1 positive reply back to the inquiring node; otherwise, there will be one message transmission to the next node of the path plus the transmissions inside the subpath P' (with $t-2$ nodes) rooted at the next node. This leads to the following recurrence:

$$\begin{aligned} m(t-1) &= 1 \times F_1 + (1 + m(t-2)) \times (1 - F_1) \\ &= am(t-2) + 1, \end{aligned}$$

where, as in (4), $m(0) = 1$, since the last node receiving the message will always reply to the inquiring node whether it knows x or not. The solution to the above recurrence is:

$$m(t-1) = \frac{a^t - 1}{a - 1},$$

which gives:

$$\overline{M_t^{(p)}} = ap + ap\frac{a^t - 1}{a - 1}. \tag{11}$$

3.5 Comparison

The three performance measures are shown in Fig. 3 for all three strategies and for a network consisting of $N = 1000$ nodes and a total of $R = 5000$ resources. Two different node degrees ($d = 4, 6$) and two different cache sizes ($k = 20, 250$) are considered. The resource in question is assumed to be offered by $n_x = 4$ nodes in total. Notice that the plots include the probability of *not* finding the resource, i.e. $1 - Q_t$ instead of Q_t.

The graphs show the random paths strategy for $p = 1$ and 4 paths. For the teeming algorithm, we chose $\phi = 0.5$; on the average half of a node's neighbors are contacted

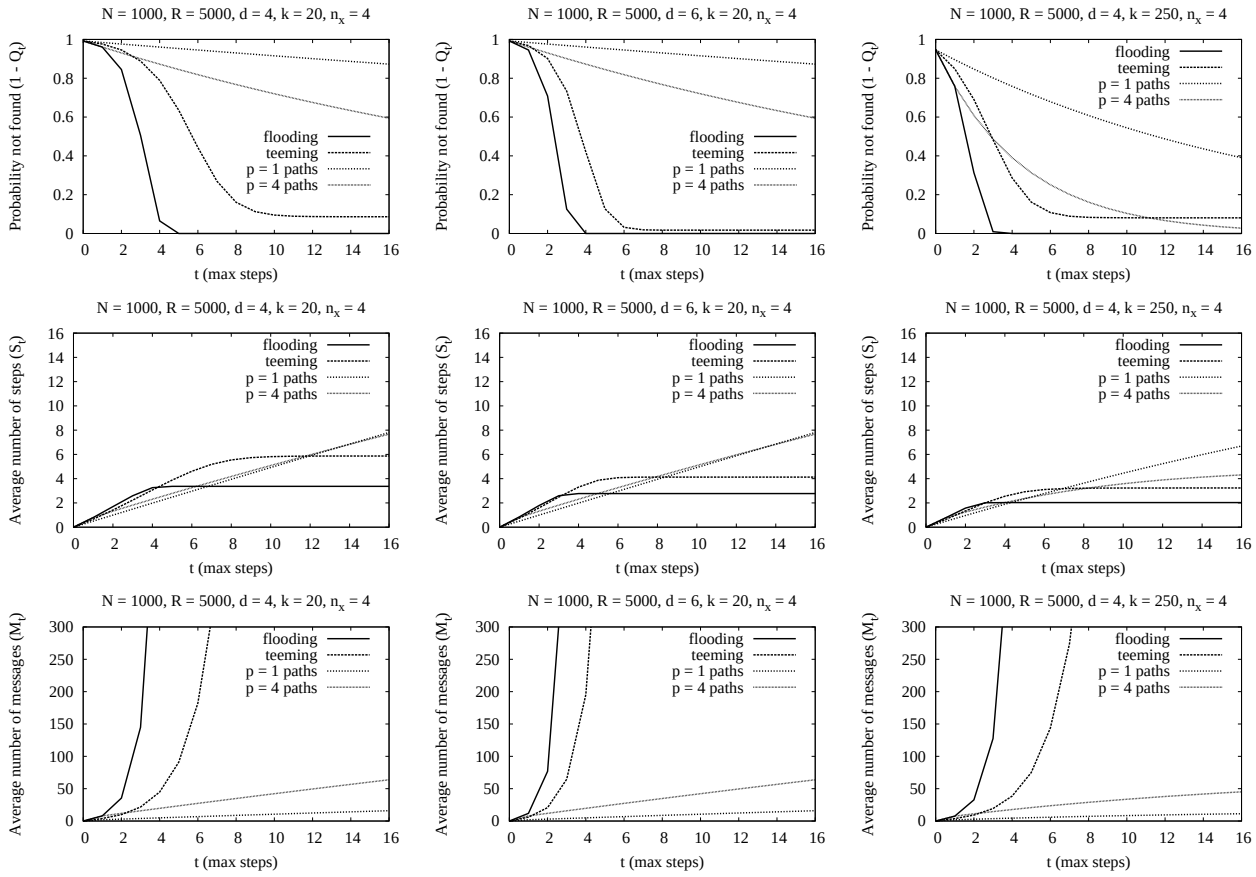

Figure 3: Comparison of the proposed algorithms: probability of not finding the resource ($= 1 - Q_t$), mean path length ($= \overline{S_t}$) and average number of message transmissions ($= \overline{M_t}$). The p-paths algorithm is shown for $p = 1, 4$. The teeming algorithm uses $\phi = 0.5$.

each time. Larger values of ϕ will yield less steps but more message transmissions as is evident from (7), (8), while for ϕ approaching 1, teeming approaches flooding.

The plots show the positive and negative aspects of the algorithms. Flooding/teeming yield lower probabilities of missing the requested resource and within a smaller number of steps, as compared to random paths. However, the number of message transmissions is excessive. Teeming constitutes possibly the better tradeoff, if the probability ϕ is chosen appropriately.

The random paths strategy performs quite poorly for very small values of p (e.g. 1 or 2), since it has a high probability of not finding the requested resource within a

limited number of steps. However, for 4 paths or more, and larger cache sizes, its performance improves substantially, although achieving comparable probabilities of success with teeming and flooding requires more steps.

Given that the resource is found, the graphs show that the average number of steps is smaller for the random paths strategy and for small values of t. The reason for this seemingly unexpected behavior is that if the resource is found, flooding and teeming will locate it at later steps since most of the contacted nodes will be contained at the last levels of the search tree. Thus, for small t, most of the time the resource will be discovered in about t steps. On the other hand, a random path will locate it on the average at the middle node, i.e. in about $t/2$ steps. However, after some point flooding and teeming will have contacted most of the nodes in the network; consequently, as t grows, both strategies need no additional steps. For example, it is easy to see that for $d = 4$, flooding will have covered all 1000 nodes in the network in less than 5 steps.

The beneficial effect of increasing the cache size is also apparent, especially for the random paths strategy, since it reduces the average number of steps and messages while increasing the probability of success. Increasing the degree of nodes is not by itself helpful for this strategy, unless the number of paths is also increased. For the other two strategies, increasing the cache size boosts the performance for the smaller values of t since, as noted above, for larger t a large number of nodes in the network will be contacted anyway; these strategies are more sensitive to the increase of nodes' degree. For example, in the considered system, the number of messages for flooding is exponential on the parameter $c = ad$. Increasing d by 50% increases c also by 50%, while increasing the cache size by a factor of 10 decreases a and c by only 2.5%.

3.6 Validation

To validate the theoretical analysis, we developed simulators (written in C) for each of the proposed strategies. The simulators, upon initialization, construct a network of N nodes, each node having d random neighbors, and assign each of the R resources to random nodes. Each resource x is assigned to n_x nodes in total. Next, the caches of all nodes are filled with random k-element subsets of the available resources where k can be anywhere from 0 to R.

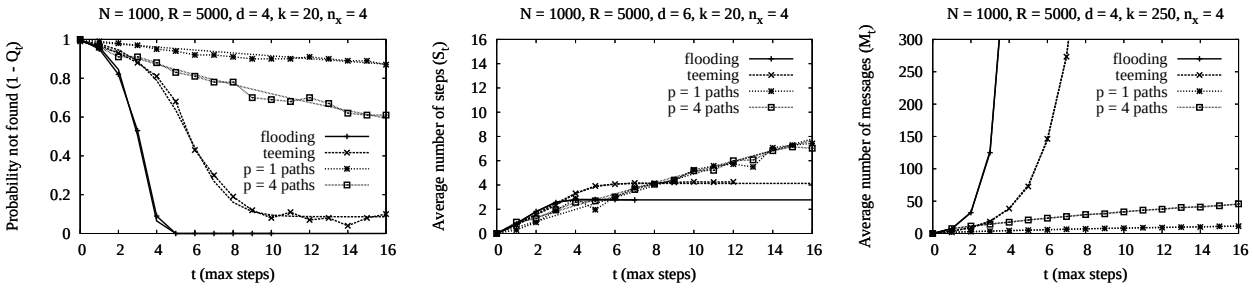

Figure 4: Simulation (patterned lines) and analytical (unpatterned lines) results.

After the initialization, simulation sessions for different values of t take place, with a random node issuing one request for a random resource x each time. Measurements obtained from each simulation session include the number of messages, the number of steps and a flag denoting whether the resource was found or not. For each of the proposed algorithms, at least 500 such sessions are performed and the accumulated results are averaged.

In Fig. 4, we give a number of simulation results for each of the proposed algorithms (patterned lines), along with the curves obtained from the theoretical analysis (unpatterned lines). The test cases are the same as in Fig. 3, i.e. there were $N = 1000$ nodes, $R = 5000$ resources and the resource was offered by $n_x = 4$ nodes. The results include the probability of not locating the resource ($= 1 - Q_t$) for the case of $d = 4$, $k = 20$, the average number of steps ($= \overline{S_t}$) for the case of $d = 6$, $k = 20$ and the average number of message transmissions ($= \overline{M_t}$) for the case of $d = 4$, $k = 250$.

It is clearly seen that the simulation results match the analytical ones quite closely. The approximation in Sections 3.2 and 3.3 produces negligible error. A slight underestimation of the probability of missing the requested resource in pure flooding shows up only for small values of the allowable number of steps (t).

4 Hot-spot analysis

It is quite common in practice that some resources are more popular than others [19]. The former are called *hot spots* and the latter *cold spots*. In this section, we extend our analysis to account for such non-uniform traffic.

Under the presence of hot spots, searching for a resource will be affected in two ways:

- a hot resource will be offered by more nodes, which means that n_x may take higher values
- the cache contents will no longer be uniformly random – hot spots / resources will be present in a higher percentage of caches than cold spots.

We will assume that within each class (hot or cold) all resources are equiprobable. In particular, each hot spot will be assumed to appear in a percentage h of all caches. Finally, we let r_h denote the fraction of resources that are hot – then $r_h R$ is the total number of hot spots, while the remaining $(1 - r_h)R$ are cold spots.

4.1 Searching for hot spots

Assume we are searching for a particular resource x. As in Section 3.1, the probability that any given node knows about it is equal to:

$$F_1 = 1 - P[\text{node does not offer } x]P[x \notin \text{node's cache}].$$

$P[\text{node offers } x] = \frac{n_x}{N}$, as we have already seen. If x is a hot spot then the probability of finding it in a cache is equal to h, which gives:

$$F_1 = 1 - (1 - h)\left(1 - \frac{n_x}{N}\right),$$

and by setting $a = (1 - h)(1 - n_x/N)$, we get $F_1 = 1 - a$.

4.2 Searching for cold spots

Assume now that the resource x we are searching for is a cold spot. Given N caches, each one holding contact information for k resources, there are in total kN cache entries in the whole network. Each hot spot appears in hN caches, and thus $r_h RhN$ entries are occupied by hot spots, in total. The rest will be occupied by cold spots, and since all of them are equiprobable, each of the cold resources appears on the average:

$$\frac{kN - r_h RhN}{R - r_h R} = \frac{N(k - r_h Rh)}{R(1 - r_h)}$$

Metric	Flooding	Teeming	Paths
Q_t	$1 - a^{\frac{d^{t+1}-1}{d-1}}$	$1 - a\left(1 - \phi Q_{t-1}\right)^d$	$1 - a^{pt+1}$
$\overline{S_t}$	$t - \frac{t}{Q_t} + \frac{1}{Q_t}\sum_{i=1}^{t} a^{\frac{d^i-1}{d-1}}$	$t - \frac{1}{Q_t}\sum_{i=0}^{t-1} Q_i$	$\frac{a-(1+t-ta^p)a^{pt+1}}{(1-a^p)(1-a^{pt+1})}$
$\overline{M_t}$	$a + \frac{(c^t-1)(2c-a)}{c-1},\ c = ad$	$a + \frac{(c^t-1)(2c-a)}{c-1},\ c = ad\phi$	$ap + ap\frac{a^t-1}{a-1}$

Search type	Value of a
Uniformly random case	$\frac{(N-n_x)(R-k)}{NR}$
Search for hot spots	$(1-h)\left(1-\frac{n_x}{N}\right)$
Search for cold spots	$\left(1-\frac{n_x}{N}\right)\left(1-\frac{k-hr_hR}{R(1-r_h)}\right)$

Table 1: Performance formulas ($t \geq 1$, $Q_0 = 1 - a$).

times, or equivalently, in a portion of

$$\frac{N(k - r_h R h)}{R(1 - r_h)} \Big/ N$$

of the caches. Consequently, we obtain:

$$F_1 = 1 - \left(1 - \frac{n_x}{N}\right)\left(1 - \frac{k - hr_hR}{R(1-r_h)}\right).$$

Setting $a = (1 - n_x/N)(1 - (k - hr_hR)/(R(1 - r_h))$, F_1 becomes equal to $1 - a$.

In conclusion, F_1, the probability that a given node knows about a resource x, is always given by $F_1 = 1 - a$. The value of a depends on the repetition of resources, the contents of the cache and the type of resource we are searching for. Apart from the different values of a, the analysis in the previous sections remains exactly the same in every case. The analytical results are summarized in Table 1.

4.3 Validation

In Fig. 5, we present simulation results (patterned lines), along the theoretical ones (unpatterned lines) for a system with $N = 1000$ nodes, $R = 5000$ resources, node degree $d = 4$ and cache size $k = 20$, for all three strategies. In the first set of experiments, we

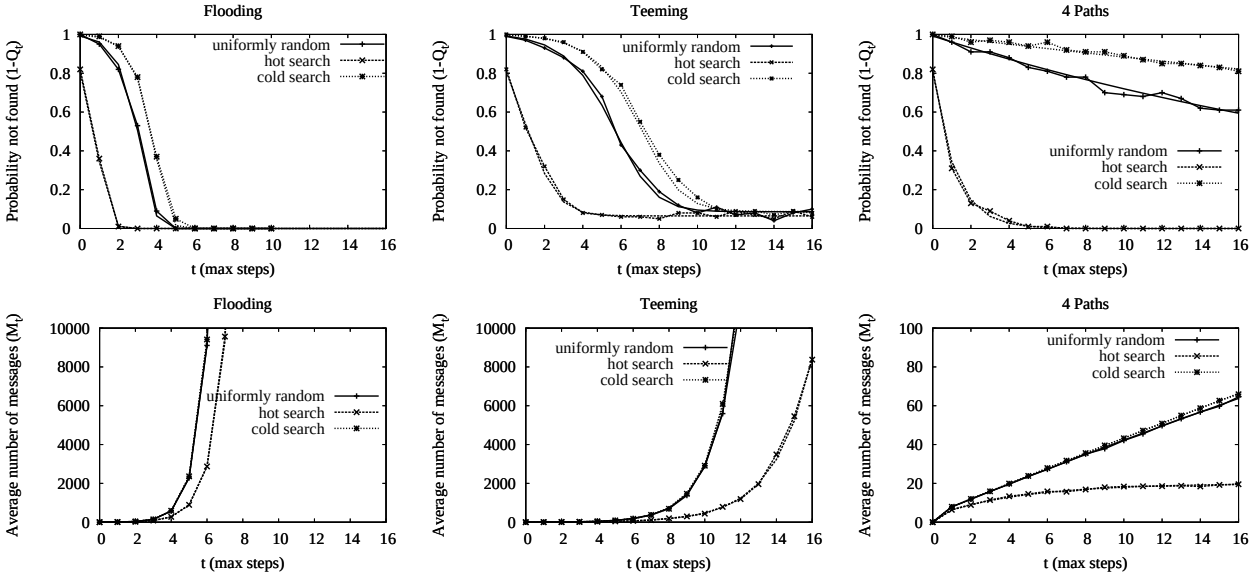

Figure 5: Uniform case, hot spot search and cold spot search: simulation (patterned lines) and analytical (unpatterned lines) results. For the uniform case $n_x = 4$, for a cold spot $n_x = 50$ and for a cold spot $n_x = 2$.

assume no hot/cold spots, and a repetition of $n_x = 4$ of the resource exactly as in Fig. 4. In the second and third sets, we consider a total of 100 (or $r_h = 2\%$) of the resources to be hot spots and that each hot spot appears in a percentage of $h = 15\%$ of all caches. Furthermore, a hot spot is assumed to be offered by $n_x = 50$ nodes, while cold spots were only offered by 2 nodes in total.

Fig. 5 shows the probability of missing the resource and the corresponding number of messages for all three cases (uniform, hot, cold) and all three strategies. Simulation and analytical results are in close agreement. As expected, the hot-spot searches are performed more efficiently, for all strategies.

5 Accounting for node unavailability

Dynamic resource sharing systems evolve over time, with new nodes being added, while some older nodes departing. This is especially true for peer-to-peer systems. For example in Gnutella, the uptime of nodes, measured as the percentage of time that the peer is available and responding to messages, is small [19]. In addition, there is always

a possibility for some nodes to malfunction or fail, thus, being unable to participate in searches. In multi-agent systems, agents may also be mobile, leaving their host environment at will. Such phenomena render some of the links in the overlay network invalid; search queries can not be propagated through them.

We model this situation by letting each node have a fixed probability $\mathcal{P}$ of being on-line. Off-line nodes are nodes that either departed, moved or malfunctioned. Queries sent to such nodes do not propagate any further. Under such a probability, on the average $(1-\mathcal{P})N$ nodes will be off-line. In this section, we update our performance metrics to account for this type of phenomena.

A search is considered successful, only if the resource is obtained, i.e. the inquiring node finds the contact information of a node that provides the required resource *and* that node is on-line. Thus, unsuccessful searches result either from failing to locate the requested resource or from correct discovery of a node offering the resource but unavailability of that node.

Before proceeding with the analysis, we note that independently of the search strategy, the probability of successfully obtaining a resource is limited by:

$$Q_t \leq 1-(1-\mathcal{P})^{n_x}. \tag{12}$$

This is the probability at least one of the n_x providers of the resource being on-line. For example, if a resource is offered by only one node, and $\mathcal{P}=60\%$, the probability of successfully obtaining the resource cannot be larger than 60%, no matter what search strategy is utilized.

Assume we search for a resource x and let $P_{\text{nof}} = P[\text{a node does not offer } x] = 1-n_x/N$, and $P_{\notin} = P[x \notin \text{a node's cache}]$ (recall that $P_{\text{nof}}P_{\notin} = a$).

During the search, queried nodes reply affirmatively (and do not propagate the search further) if they either offer x or have it cached; in the latter case the reply will be invalid with probability $1-\mathcal{P}$ of the provider of the cached resource being off-line. Consequently, the probability p_c (p_w) that a node replies correctly (wrongly) is:

$$\begin{aligned} p_c &= (1-P_{\text{nof}}) + P_{\text{nof}}(1-P_{\notin})\mathcal{P} &&= \frac{n_x}{N} + (1-\frac{n_x}{N}-a)\mathcal{P}. \\ p_w &= P_{\text{nof}}(1-P_{\notin})(1-\mathcal{P}) &&= 1-a-p_c. \end{aligned}$$

The search will be propagated to other nodes only if the current node does not offer the resource and does not have it cached. This probability is equal to $P_{\text{nof}}P_{\notin} = a$.

5.1 Flooding and teeming

To derive the probability Q_t of locating a resource x, we proceed in a manner similar to that of Sections 3.2 and 3.3. We only consider the teeming algorithm recalling that if the probability of contacting a neighbor is $\phi = 1$ we have in effect a pure flooding strategy.

If the search is unsuccessful (probability $1-Q'_t$), it is due to either one of the following facts:

- the inquiring node knows about an off-line resource provider (with probability p_w) or,
- the inquiring node propagates the search (with probability a) and none of the d "subtrees" unfolding from the inquiring node's neighbors replies with a successful answer. Such a subtree has $t-1$ levels; it sends a successful reply only if it is contacted by the inquiring node (with probability ϕ), its root node is on-line (with probability $\mathcal{P}$) and locates the requested resource successfully (with probability Q'_{t-1}).

Thus, the probability of not finding x through the search procedure is given by the following recurrence:

$$1 - Q'_t = p_w + a\left(1 - \mathcal{P}\phi Q'_{t-1}\right)^d,$$

which gives:

$$Q'_t = a + p_c - a\left(1 - \mathcal{P}\phi Q'_{t-1}\right)^d.$$

with $Q'_0 = p_c$. Taking into account (12), we obtain:

$$Q_t^{(T)} = \min\{Q'_t, 1 - (1-\mathcal{P})^{n_x}\}.$$

The average number of steps is given by Eq. (1). The number of messages is given by the recurrence:

$$\overline{M_t^{(T)}} = a(d\phi + d\phi\mathcal{P}m(t-1)),$$

where the inquiry is transmitted to $d\phi$ of the root's neighbors[†] and $m(t-1)$ is the transmissions within the subtrees T with $t-1$ levels rooted at the contacted neighbors.

[†]Notice that we count all $d\phi$ transmissions even if some of them are sent to unavailable (off-line) neighbors

For such a T, further message transmissions take place only if its root node is on-line ($\mathcal{P}$). In such a case, if x is found in the subtree's root, there will be 1 positive reply back to the root. Otherwise:

$$m(t-1) = 1 \times (1-a) + (d\phi + d\phi\mathcal{P}m(t-2)) \times a$$

$$= ad\phi\mathcal{P}m(t-2) + ad\phi + 1 - a,$$

with boundary $m(0) = 1$. Solving the recurrence and simple manipulations yield:

$$\overline{M_t^{(T)}} = a + \frac{(c^t - 1)\,(c + c/\mathcal{P} - a)}{c-1}, \quad c = ad\phi\mathcal{P}.$$

5.2 Random paths

Similarly to Sections 5.1 and 3.4, the probability of an unsuccessful search is given by:

$$1 - Q'_t = p_w + a(1 - \mathcal{P}q_{t-1})^p,$$

where q_{t-1} is the probability of finding x in one (of the p) random paths, within the remaining $t-1$ steps. Such a path is traversed with probability $\mathcal{P}$ of its first node being on-line. Locating x in such a path means that its first node replies correctly or propagates the search and the resource is successfully found at later steps, which leads to the following recurrence:

$$q_{t-1} = p_c + a\mathcal{P}q_{t-2},$$

with $q_0 = p_c$.

If an off-line intermediate node is selected in any of the paths, the search in that path stops. This deteriorates the performance of the strategy, especially for small values of p. We thus consider the following variation. Instead of randomly selecting among all neighbors, each node selects randomly among its neighbors *that are on-line*. Adopting this policy, q_{t-1} will now be given by:

$$q_{t-1} = p_c + a(1 - (1-\mathcal{P})^d)q_{t-2},$$

where $u = 1 - (1-\mathcal{P})^d$ is the probability of having at least one on-line neighbor. The solution is $q_{t-1} = p_c((au)^t - 1)/(au - 1)$, which gives:

$$Q'_t = a + p_c - a\left(1 - p_c\mathcal{P}\frac{(a - a(1-\mathcal{P})^d)^t - 1}{a - a(1-\mathcal{P})^d - 1}\right)^p.$$

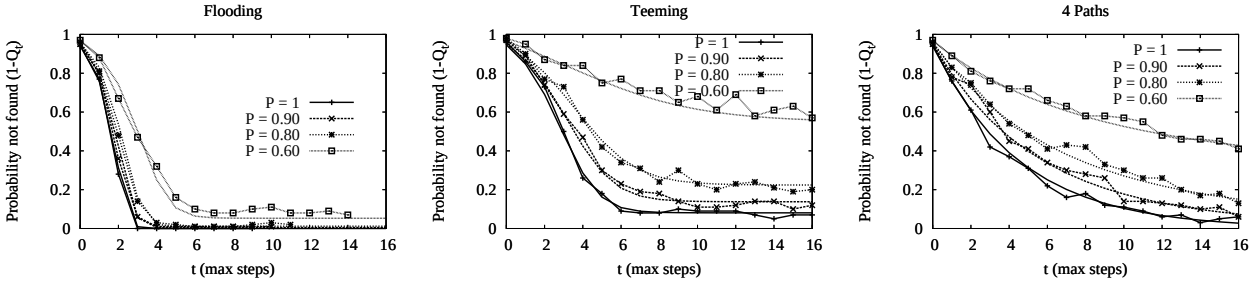

Figure 6: Simulation (patterned lines) and analytical (unpatterned lines) for the case of nodes being on-line with probability $\mathcal{P}$.

Using (12), we finally obtain:

$$Q_t^{(p)} = \min\{Q'_t, 1 - (1 - \mathcal{P})^{n_x}\}.$$

The average number of steps is given by Eq. (1), while the number of messages is given by:

$$\overline{M_t^{(p)}} = a(p + p\mathcal{P}m(t-1)),$$

where $m(t-1)$ is the transmissions occurring within a particular path of $t-1$ nodes. As in Section 3.4, we arrive at the following recurrence:

$$m(t-1) = aum(t-2) + 1,$$

which has $m(0) = 1$ and a solution of $m(t-1) = ((au)^t - 1)/(au - 1)$. Consequently,

$$\overline{M_t^{(p)}} = ap + ap\mathcal{P}\frac{(a - a(1-\mathcal{P})^d)^t - 1}{a - a(1-\mathcal{P})^d - 1}.$$

5.3 Validation

To validate our theoretical analysis, the simulator was modified to account for unavailable nodes. Before each simulation round, each node is marked as on-line with a given probability $\mathcal{P}$. During the simulation sessions, any replies which point to owners that are off-line are considered wrong. A search is considered successful only if it results in locating a resource owner and that node is on-line.

In Fig. 6, we present simulation results (patterned lines), along the theoretical ones (unpatterned lines) for a system with $N = 1000$ nodes, $R = 5000$ resources, node degree

$d = 4$ and cache size $k = 250$, for all three strategies. We present the probability of successfully obtaining the requested resource for $\mathcal{P} = 100\%, 90\%, 80\%, 60\%$.

Simulation and analytical results are in close agreement. The variation of random paths (where each of the p paths continues to evolve as long as intermediate nodes have at least one on-line neighbor) performs comparably with teeming when the percentage of on-line nodes is small. In this case, during teeming, nodes contact successfully quite few neighbors, since only a few nodes are on-line and among those, only a portion ϕ is selected. In our example, for $\mathcal{P} = 60\%$, $\phi = 0.5$ and $d = 4$, each node will have about 2.4 on-line neighbors, and contacts on average 1.2 of them.

6 Related work and applications of flooding

There are many applications of flooding based search in distributed systems. We discuss next related work on flooding-based search in three different settings, namely peer-to-peer systems, multi-agent systems and distributed networks.

Resource discovery in peer-to-peer systems Recently, peer-to-peer (p2p) computing has attracted much attention as a new distributed computing paradigm of sharing resources available at the edges of the network. A p2p system is a fully-distributed cooperative network in which nodes collectively form a system without any supervision. An issue central to p2p systems is discovering a peer that offers a particular resource. There are two types of p2p systems depending on the way resources are located in the network. In *structured p2p systems*, resources are placed at peers at specified locations. Most resource discovery procedures in structured p2p systems (such as CAN [17], Chord [22] and Past [18]) build a distributed hash table. With distributed hashing, each resource is associated with a key and each node (peer) is assigned a range of keys. In *unstructured p2p* systems, resources are located at random points. In this context, flooding-based approaches to resource discovery have been proposed, in which each peer searching for a resource contacts all its neighboring peers. Gnutella [8] is an example of such an approach. Unstructured approaches distribute the load and increase tolerance to failures.

The analysis in this paper is applicable in the case of fully-decentralized, unstructured

search. While there have been empirical studies (e.g. [19]) and some simulation-based analysis (e.g. [15]) of flooding and its variants for p2p systems, analytical results are lacking. Here, we analytically evaluate various alternatives of flooding-based approaches.

Open multi-agent systems Another application of flooding-based search is in multi-agent systems. A multi-agent system (MAS) is a loosely coupled network of software agents that cooperate to solve problems that may be beyond the individual capacities or knowledge of each particular agent. In a MAS, computational resources are distributed across a network of interconnected agents. To fulfill their goals, agents in a MAS need to use resources provided by other agents. To use a resource, an agent must contact the agent that provides it. However, in an *open* MAS, an agent does not know which agents provide which resources. Furthermore, it does not know which other agents participate in the system. A common approach to the resource discovery problem is to introduce middle agents or directories that maintain information about which agents provide which resources [23]. Thus to find a resource, an agent has first to contact the middle agent. However, middle agents can become bottlenecks and contradict the distribution goals set by a MAS along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, and flexibility.

An approach based on flooding is applicable to the resource discovery problem in open MAS [6, 5]. In this approach, each agent maintains a limited size local cache with the contact information for k different resources (i.e., for each of the k resources, one agent that offers it). This results in a fully distributed directory scheme, where each agent stores part of the directory. The agents in the cache of an agent A are the agents that A knows about. We call them the agent's neighbors.

The only other performance studies of the use of local caches for resource location in MAS that we are aware of are [4] and [20]. In [4], a depth first traversal of what corresponds to our overlay network is proposed. Experimental results are presented that show that this approach is more efficient in terms of the number of messages than flooding for particular topologies, in particular, the ring, star, and complete graph topologies. There are no analytical results. In [20], the complexity of the very limited case of lattice-like graphs (in which each agent knows exactly four other agents in such a way that a static grid is formed) is analyzed.

Resource discovery in distributed networks The problem that we study in this paper can be seen as a variation of the resource discovery problem in distributed networks, where nodes in a large, dynamic communication infrastructure know only about a small number of other nodes and wish to discover all nodes currently existing in the system. The problem was initially introduced in [10], deterministic algorithms with improved complexity were presented in [13], and the case of asynchronous networks was studied in [12, 2]. However, there are important differences: (i) we are interested in learning about one specific resource as opposed to learning about all other known nodes, (ii) our network may be disconnected and (iii) in our case, each node has a limited-size cache, so at each instance, it knows about at most k other nodes. Finally, flooding has also been used in ad-hoc routing (e.g. [9, 1]). In this case, the objective is to ensure that a message starting from a source node reaches its destination.

7 Conclusions

In this paper, we consider the general problem of discovering resources in a distributed environment. In particular, we study the performance of a number of flooding-based approaches to this problem. With flooding, a node searching for a resource contacts its neighbors, which in turn contact their own neighbors and so on until a node with the requested resource is located. Flooding assumes no knowledge about the network topology and the distribution of resources thus offering an attractive method for resource discovery in dynamically evolving networks. However, pure flooding incurs large network overheads. To address this, we consider two variations, namely teeming and random paths, that confine the search space. We derive analytical results for the performance of each strategy which are validated through simulation. Our results take into account non-uniform resource requests as well as node unavailability.

As future research, we plan to study the performance of other resource discovery strategies. An interesting one is a refinement of teeming, called teeming with decay [14]. In this policy, ϕ (the probability of selecting a neighbor to propagate a request) decreases as the search progresses, so as to avoid the excessive number of messages in the later steps of the original algorithm.

Because the overlay network may contain directed cycles, queries may reach interme-

diate nodes through multiple paths. Those nodes will propagate the same query multiple times, creating unnecessary duplicate messages, which cause a high traffic overhead [15] while they do not increase the probability of successful resource discovery. Duplication detection mechanisms should thus be present at each node. Our research plans include extending our analysis to systems with duplicate message avoidance.

References

[1] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," *Ad Hoc Networks*, vol. 2, no. 1, pp. 1–22, Jan. 2004.

[2] I. Abraham and D. Dolev, "Asynchronous Resource Discovery," in *Proc. PODC03, 22nd ACM Symposium on Principles of Distributed Computing*, 2003, pp. 143–150.

[3] F. Banaei-Kashani and C. Shahabi, "Criticality-based Analysis and Design of Unstructured Peer-to-Peer Networks as Complex Systems," in *Proc. GP2PC 2003, 3rd Int'l Workshop on Global and Peer-to-Peer Computing*, 2003.

[4] M. A. Bauer and T. Wang, "Strategies for Distributed Search," in *Proc. CSC '92, ACM Computer Science Conference*, 1992, pp. 251–260.

[5] V. V. Dimakopoulos and E. Pitoura, "A Peer-to-Peer Approach to Resource Discovery in Multi-Agent Systems," in *Proc. CIA '03, 7th Int'l Workshop on Cooperative Information Agents*, 2003, pp. 62–77.

[6] V. V. Dimakopoulos and E. Pitoura, "Performance Analysis of Distributed Search in Open Agent Systems," in *Proc. IPDPS '03, Int'l Parallel and Distributed Processing Symposium*, 2003.

[7] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," in *Proc. 10th ACM SIGCOMM Conference*, 1998.

[8] Gnutella RFC, "http://rfc-gnutella.sourceforge.net."

[9] Z. Haas, J. Y. Halpern, and L. Li, "Gossip-Based Ad Hoc Routing," in *Proc. IEEE INFOCOM 2002*, 2002, pp. 1707–1716.

[10] M. Harchol-Balter, T. Leighton, and D. Lewin, "Resource Discovery in Distributed Networks," in *Proc. PODC '99, 18th ACM Symposium on Principles of Distributed Computing*, 1999, pp. 229–337.

[11] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," in *11th ACM CIKM Conference*, 2002.

[12] S. Kutten and D. Peleg, "Asynchronous Resource Discovery in Peer to Peer Networks," in *Proc. SRDS02, 21st IEEE Symposium on Reliable Distributed Systems*, 2002.

[13] S. Kutten, D. Peleg, and U. Vishkin, "Deterministic Resource Discovery in Distributed Networks," in *Proc. SPAA01, 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, 2001, pp. 77–83.

[14] E. Leontiadis, V. V. Dimakopoulos, and E. Pitoura, "Cache Updates in a Peer-to-Peer Network of Mobile Agents," in *4th International Conference on Peer-to-Peer Computing*, 2004.

[15] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," in *Proc. ICS2002, 16th ACM Int'l Conference on Supercomputing*, 2002, pp. 84–95.

[16] E. P. Markatos, "Tracing a Large-Scale Peer to Peer System: an Hour in the Life of Gnutella," in *Proc. CCGrid 2002, 2nd IEEE Int'l Symposium on Cluster Computing and the Grid*, 2002, pp. 65–74.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," in *Proc. ACM SIGCOMM '01 Conference*, 2001, pp. 161–172.

[18] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility," in *Proc. SOSP 2001, 18th ACM Symp. on Operating System Priciples*, 2001, pp. 188–201.

[19] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proc. MMCN '02, Multimedia Computing and Networking 2002*, 2002.

[20] O. Shehory, "A Scalable Agent Location Mechanism," in *Proc. ATAL '99, 6th Int'l Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages*, ser. LNCS, vol. 1757. Springer, 2000, pp. 162–172.

N	number of nodes in the network
R	number of different resources
d	the (out-)degree of each node
k	cache size per node
x	the resource we search for
n_x	repetitions of resource x
t	maximum allowable number of search steps
r_h	portion of resources that are hot spots
h	portion of nodes that have cached a given hot spot
$\mathcal{P}$	portion of nodes that are on-line
F_1	probability that a resource is known to a given node
a	$= 1 - F_1$, probability a resource is *not* known to a node
Q_t	probability of locating a resource within t steps
$\overline{S_t}$	average number of steps needed to locate a resource
$\overline{M_t}$	average number of message transmissions.

Table 2: Notation

[21] K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability," in *O'Reilly's www.openp2p.com site*, 2001.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. ACM SIGCOMM '01 Conference*, 2001, pp. 149–160.

[23] K. Sycara, M. Klusch, S. Widoff, and J. Lu, "Dynamic Service Matchmaking Among Agents in Open Information Environments," *SIGMOD Record*, vol. 28, no. 1, pp. 47–53, March 1999.

A Notation

The notation used in this paper is summarized in Table 2. The top portion of the table contains the given parameters and the bottom half contains the derived quantities.